A Bayesian Approach to Predicting NFL Quarterback Scores in Fanduel Tournaments

STAT 578, Fall 2017, ***Team 5****: Aaron Ray, Kiomars Nassiri, Michael Chan*

October 25, 2017

Project Description

The National Football League (NFL), being one of the major professional sports leagues in North America, has a wide audience. participates in the NFL craze by competing in fantasy football tournaments organized by the daily fantasy site, "FanDuel.com". Participants in a **Fantasy Football** game act as the managers of a virtual footbal team and try to maximize their points by picking up the best line-up. Points are given based on actual performance of players in real-world competition. For the purpose of this project we have chosen to work with the data gathered from the **FanDuel** internet company. We will leverage a Hierarchical Bayesian approach with the Markov Chain Monte Carlo method to predict the fantasy points likely to be scored by an NFL quarterback in any given game. The goal is to predict the points scored by each player given certain prior conditions and predictor variables that will assist our model in providing credible posterior prediction intervals.

The analysis is inspired by the study presented in the article, **Bayesian Hierarchical Modeling Applied to Fantasy Football Projections for Increased Insight and Confidence**, by Scott Rome.

Team Members

- **Aaron Ray** (aaronwr2@illinois.edu)*
- **Kiomars Nassiri** (nassiri2@illinois.edu)
- **Michael Chan** (mhchan3@illinois.edu)

*Contact Person

Dataset Description

Team has set up a process to gather the historical data from the RotoGuru website. The following is the code used to get the data from RotoGuru:

```
# Scrape rotoguru1 site for weekly FanDuel stats and bind each week's data to the
# pre-defined dataframe, 'd'.

for(year in 2014:2017){
  for(week in 1:16){
    page = read_html(
      gsub(" ","",
           paste("http://rotoguru1.com/cgi-bin/fyday.pl?week=",week,"&year=",
                 year ,"&game=fd&scsv=1")
      ))
    dtext = page %>% html_nodes("pre") %>% html_text(trim = TRUE)
    dtable = read.table(text=dtext, sep = ";", header=TRUE, col.names = cnames,
```

```
                        quote=NULL)
    d = rbind(d,dtable)
  }
}
```

Data cleaning is performed using R routines. Some data cleaning tasks are needed to calculate Player rank.

Response Variables

- `FanDuelPts`: Points position at the end of a single game

Predictor Variables

- `AvgPts5Wks`: The 5 game average points of the player
- `AvgOppPAP7Wks` : The 7 game average Opposing Points Allowed to Position (OppPAP) by the current player's opposing defense. For example, if the Buffalo Bills defense allowed a total of 30 points per game to wide receivers for six games straight, then this number would equal to the average of 30 for any wide receiver facing the Bills defense.
- `Position`: The position the player plays
- `HomeGame`: Whether it is home game.
- `Rank`: The rank of a player based on recent performance

Analysis Ideas

Model

At the lowest level, we model the performance (`FanDuelPts`) as normally-distributed around a true value:

$y|\alpha, \beta_{defense}, \beta_{home}, \beta_{away}, \sigma_r^2 \sim N(\alpha + X_{defense}.\beta_{defense} + X_{home}.\beta_{home} + X_{away}.\beta_{away}, \sigma_y^2 I)$

where

α = The average fan duel point of the previous 5 weeks of the player, `AvgPts5Wks`

$\beta_{defense,p}$ = defense coefficient against team t for position p

$\beta_{home,p,r}$ = home coefficient for position p and a rank r player

$\beta_{away,p,r}$ = Away coefficient for position p and a rank r player

y = `FanDuelPts`

x_p = interaction indicator term for opposing team score allowed by position p

$x_{home,p,r}$ = interaction indicator term for rank r, position p, and whether it is home game

At higher level, we model the defense effect, $\beta_{defense}$, as how good(bad) a particular team's defense is against the player's position. We pool the effect based on the position of the player. That is, the defense coefficient is normally distributed from the same position specific distribution.

$\beta_{defense,p} \sim N(\delta_p, \sigma_\delta^2)$

where σ_δ is constant = 1000

For the home and away game effect, β_{home} and β_{away}, we model the effect for player of the same rank has the same distribution. We model the home and away game effect to be the same for players of the same position.

$\beta_{home,p,r} \sim N(\eta_r, \sigma_\eta^2)$

$\beta_{away,p,r} \sim N(\rho_r, \sigma_\rho^2)$

where σ_η, σ_ρ are constant $= 1000$

We will approximate non informative prior using:

$\sigma_y \sim Inv - gamma(0.0001, 0.0001)$

$\delta \sim N(0, 10000^2)$

$\eta \sim N(0, 10000^2)$

$\rho \sim N(0, 10000^2)$

Here is the JAGS model:

```
#sink("fdp.bug")
#cat("
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(alpha[i] + inprod(X.defense[i, ], beta.defense)
                 + inprod(X.home[i, ], beta.home)
                 + inprod(X.away[i, ], beta.away), sigmasqinv)
  }

  # In our model, we pool the beta.defense based on position.
  # i.e. All defense effects of the same position are drawn from the same distribution
  for (p in 1:Num.Position) {
    beta.defense[p] ~ dnorm(delta[p], 1/1000^2)
    delta[p] ~ dnorm(0, 1/10000^2)
  }

  # The entry of the beta.home and beta.away corresponds to Rank:Position
  # In our model, we pool the beta.home/away based on rank
  for (r in 1:Num.Rank) {
    for (t in 1:Num.Position) {
      beta.home[(r-1) * Num.Position + t] ~ dnorm(eta[r], 1/1000^2)
      beta.away[(r-1) * Num.Position + t] ~ dnorm(rho[r], 1/1000^2)
    }
    eta[r] ~ dnorm(0, 1/10000^2)
    rho[r] ~ dnorm(0, 1/10000^2)
  }

  sigmasqinv ~ dgamma(0.0001, 0.0001)
  sigmasq <- 1/sigmasqinv
}
#    ",fill = TRUE)
#sink()
```

Sample Data

```
fdp <- read.csv("fdpfinal.csv", sep = ',', header = TRUE)

head(fdp)
```

```
##   Position Year YearWeek Opponent Week PlayerId             Name    Team
## 1       QB 2015   201513 Steelers   13     1060 Hasselbeck, Matt   Colts
## 2       QB 2015   201514  Jaguars   14     1060 Hasselbeck, Matt   Colts
## 3       QB 2015   201515   Texans   15     1060 Hasselbeck, Matt   Colts
## 4       QB 2015   201516 Dolphins   16     1060 Hasselbeck, Matt   Colts
## 5       QB 2015   201501   Ravens    1     1081  Manning, Peyton Broncos
## 6       QB 2015   201502   Chiefs    2     1081  Manning, Peyton Broncos
##   HomeGame FanDuelPts FanDuelSalary AvgOppPAP7Wks SdOppPAP7Wks OallAvgPAP
## 1        0       6.86          6500         20.95        8.045      17.44
## 2        0       9.08          6600         24.16        6.738      17.44
## 3        1       8.98          6400         16.36        9.690      17.44
## 4        0       3.96          6000         20.46        6.002      17.44
## 5        1       5.90          9100         18.99       11.697      17.44
## 6        0      21.24          8200         13.85        4.342      17.44
##   OallStdevPAP AvgPts5Wks StdevPts5Wks OffRnk5Wks DefRnk7Wks
## 1        2.909      14.85        4.824      Rank1      Rank2
## 2        2.909      14.82        4.897      Rank1      Rank2
## 3        2.909      13.56        5.490      Rank1      Rank3
## 4        2.909      12.11        5.566      Rank1      Rank2
## 5        2.909      14.96        8.496      Rank1      Rank2
## 6        2.909      10.53        5.011      Rank2      Rank3
```

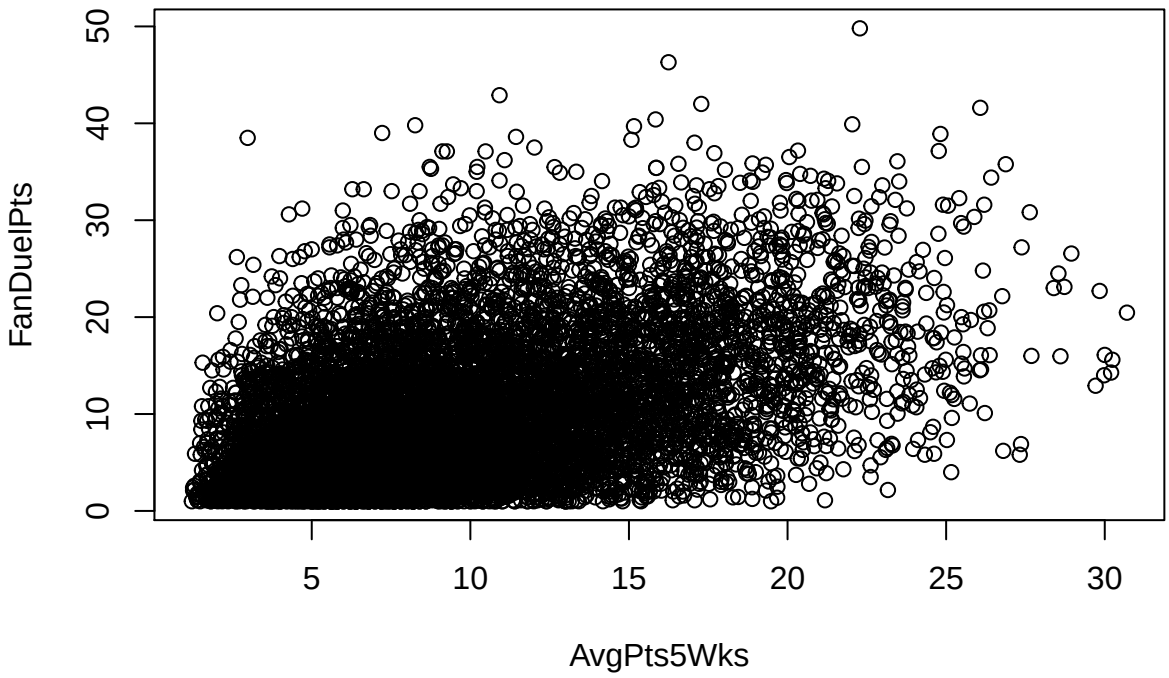
```
fdp['Rank'] = fdp$OffRnk5Wks
fdp['Locality'] = 'Away'
fdp[fdp$HomeGame == 1, 'Locality'] = 'Home'
```

Simple Ideas

$y|\alpha \sim N(\alpha, \sigma_y^2 I)$

```
mod.classic = lm(FanDuelPts ~ AvgPts5Wks, data = fdp)

plot(FanDuelPts ~ AvgPts5Wks, data = fdp)
```



All over the place, let's add the team defense

`X.defense` is the indicator matrix

** Set up train data **

```
fdp_train=fdp[fdp$Year == 2015 & fdp$Position=="QB", ]
fdp_train = droplevels(fdp_train)

Num.Opponent = length(unique(fdp_train[, "Opponent"]))
Num.Position = length(unique(fdp_train[, "Position"]))
Num.Rank = length(unique(fdp_train[, "Rank"]))

if (Num.Position == 1) {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank , data=fdp_train)
} else {
  #X.offense = model.matrix(~ 0 + AvgPts5Wks:Position, data=fdp_train)
  X.defense = model.matrix(~ 0 + AvgOppPAP7Wks:Position, data=fdp_train)
  X.home = model.matrix(~ 0 + Rank:Position , data=fdp_train)
  X.away = model.matrix(~ 0 + Rank:Position , data=fdp_train)
}


X.home = X.home * fdp_train$HomeGame
```

```r
X.away = X.away * (1- fdp_train$HomeGame)
X = cbind(X.defense, X.home, X.away)
```

```r
library(rjags)
set.seed(20171008)
```

```r
# Initialization List for the 4 chains
jags.inits=list(
  list( sigmasqinv=    0.01,  delta = -10000,
        eta = c(10000, -10000, 10000),
        rho = c(-10000, 10000, -10000),
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 ),
  list( sigmasqinv=    0.01,  delta =  10000,
        eta = c(10000, -10000, -10000),
        rho = c(-10000, 10000, 10000),
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 1 ),
  list( sigmasqinv=0.000001,  delta = -10000,
        eta = c(-10000, 10000, -10000),
        rho = c(10000, -10000, 10000),
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 2 ),
  list( sigmasqinv=0.000001,  delta =  10000,
        eta = c(-10000, 10000, 10000),
        rho = c(10000, -10000, -10000),
        .RNG.name = "base::Mersenne-Twister", .RNG.seed = 20171008 + 3 )
)

data.jags <- list(
  y= fdp_train$FanDuelPts,
  alpha = fdp_train$AvgPts5Wks,
  X.defense = X.defense,
  X.home = X.home,
  X.away = X.away,
  Num.Position=Num.Position,
  #Num.Opponent=Num.Opponent,
  Num.Rank=Num.Rank
)
```

```r
burnAndSample = function(m, N.burnin, N.iter, show.plot, mon.col) {
  update(m, N.burnin) # burn-in

  x <- coda.samples(m, mon.col, n.iter=N.iter)

  if(show.plot) {
    plot(x, smooth=FALSE)
  }

  gelman.R = gelman.diag(x, autoburnin=FALSE, multivariate = FALSE)
  print(gelman.R)

  result <- list(
    coda.sam = x,
    gelman.R.max=max(gelman.R$psrf[, 1])
  )
```

```
  return(result)
}
```

```
runModel=TRUE
runSample=TRUE

mon.col <- c("delta", "eta", "rho", "beta.defense", "beta.home", "beta.away", "sigmasq")

NSim = 50000
if (runModel) {
  m <- jags.model("fdp.bug", data.jags, inits = jags.inits, n.chains=4, n.adapt = 1000)
  save(file="fdp.jags.model.Rdata", list="m")
} else {
  load("fdp.jags.model.Rdata")
  m$recompile()
}
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 436
##    Unobserved stochastic nodes: 15
##    Total graph size: 5666
##
## Initializing model
```

```
load.module("dic")
```

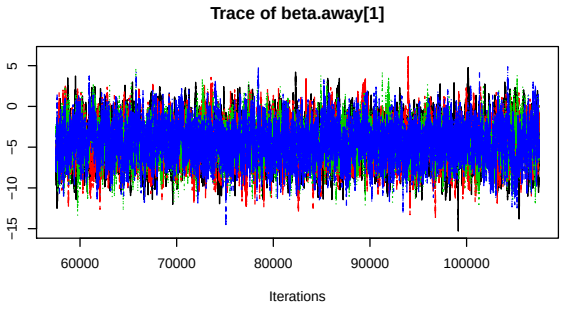
```
## module dic loaded
```

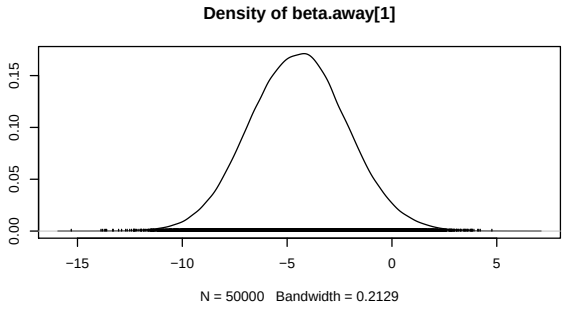
```
N.Retry.Loop = 2
if (runSample) {
  N.burnin=2500/2
  for (loopIdx in 1:N.Retry.Loop) {
    (start_time <- Sys.time())
    (N.burnin = N.burnin * 2)
    result = burnAndSample(m, N.burnin, NSim, show.plot=FALSE, mon.col = mon.col)
    (end_time <- Sys.time())
    (result$gelman.R.max)
  }
  save(file="fdp.jags.samples.Rdata", list="result")
  save(file="fdp.jags.model.Rdata", list="m")
} else {
  load("fdp.jags.samples.Rdata")
  N.burnin=2500/2 * (2**N.Retry.Loop)
}
```

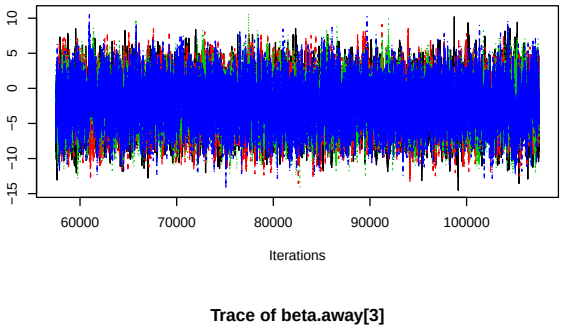
```
## Potential scale reduction factors:
##
##              Point est. Upper C.I.
```

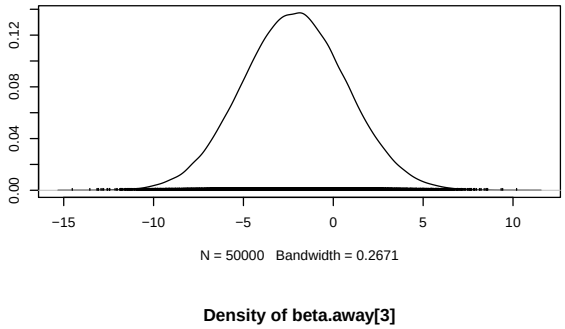
```
## beta.away[1]          1          1
## beta.away[2]          1          1
## beta.away[3]          1          1
## beta.defense          1          1
## beta.home[1]          1          1
## beta.home[2]          1          1
## beta.home[3]          1          1
## delta                 1          1
## eta[1]                1          1
## eta[2]                1          1
## eta[3]                1          1
## rho[1]                1          1
## rho[2]                1          1
## rho[3]                1          1
## sigmasq               1          1
##
## Potential scale reduction factors:
##
##              Point est. Upper C.I.
## beta.away[1]          1          1
## beta.away[2]          1          1
## beta.away[3]          1          1
## beta.defense          1          1
## beta.home[1]          1          1
## beta.home[2]          1          1
## beta.home[3]          1          1
## delta                 1          1
## eta[1]                1          1
## eta[2]                1          1
## eta[3]                1          1
## rho[1]                1          1
## rho[2]                1          1
## rho[3]                1          1
## sigmasq               1          1
```

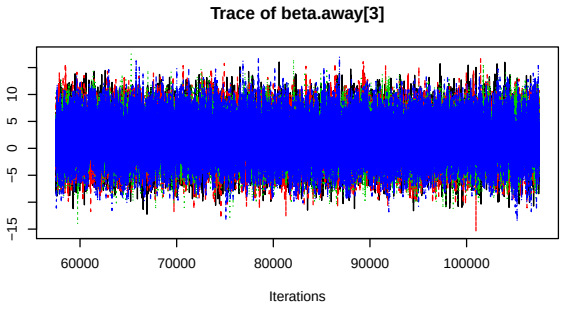
```
plot(result$coda.sam, smooth=FALSE)
```

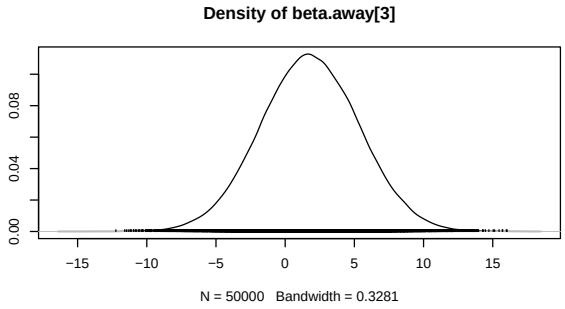

Trace of beta.away[1]
Iterations

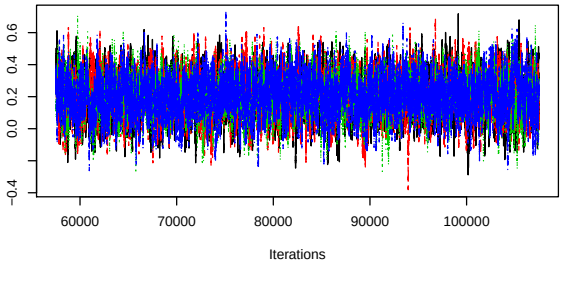

Density of beta.away[1]
N = 50000 Bandwidth = 0.2129

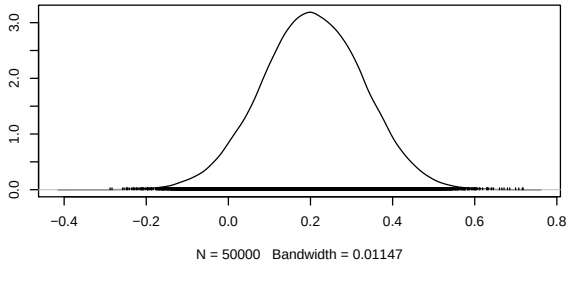

Trace of beta.away[2]
Iterations

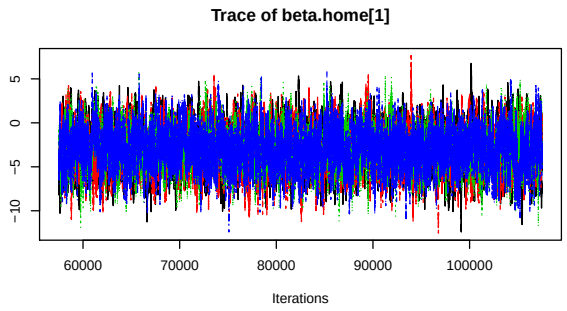

Density of beta.away[2]
N = 50000 Bandwidth = 0.2671

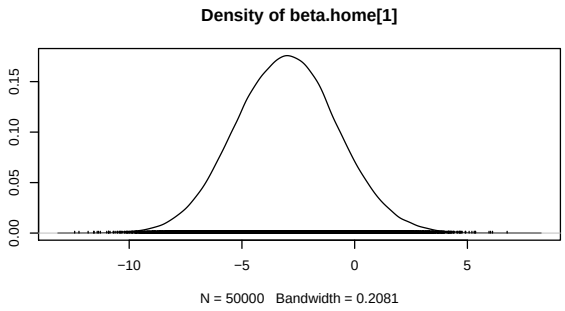

Trace of beta.away[3]
Iterations

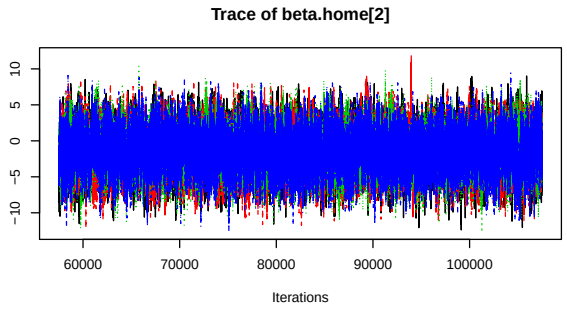

Density of beta.away[3]
N = 50000 Bandwidth = 0.3281

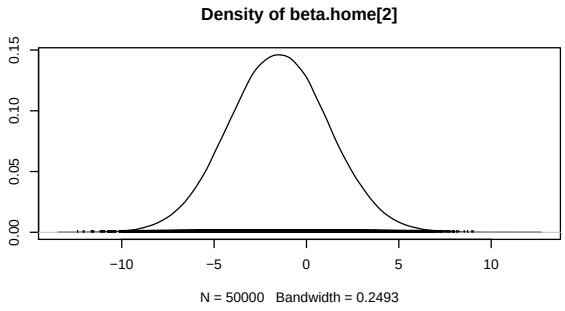

Trace of beta.defense
Iterations

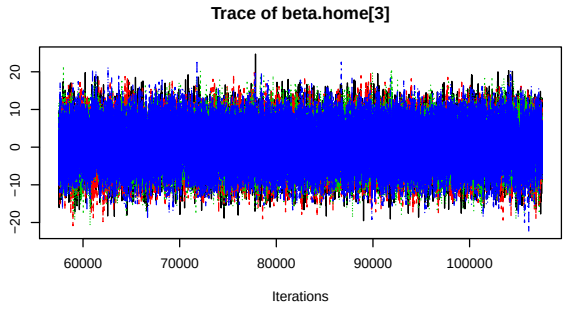

Density of beta.defense
N = 50000 Bandwidth = 0.01147

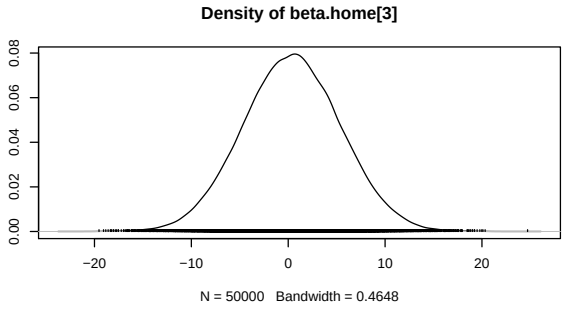

Trace of beta.home[1]
5
0
−5
−10
60000
70000
80000
90000
100000
Iterations

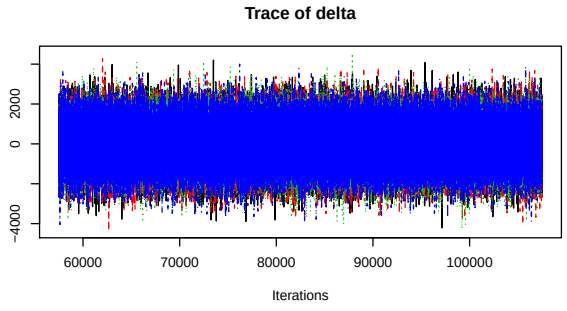

Density of beta.home[1]
0.15
0.10
0.05
0.00
−10
−5
0
5
N = 50000 Bandwidth = 0.2081

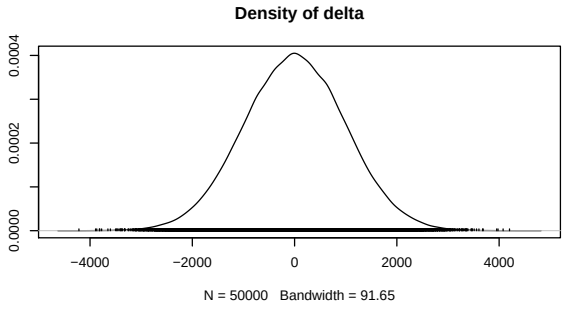

Trace of beta.home[2]
10
5
0
−5
−10
60000
70000
80000
90000
100000
Iterations

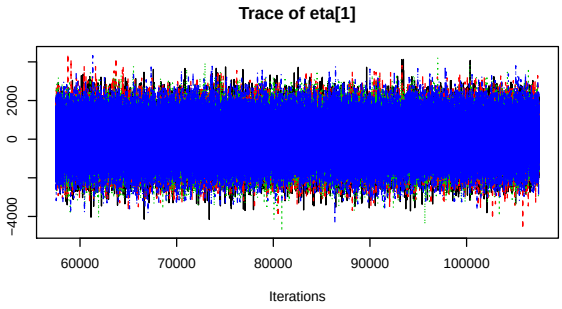

Density of beta.home[2]
0.15
0.10
0.05
0.00
−10
−5
0
5
10
N = 50000 Bandwidth = 0.2493

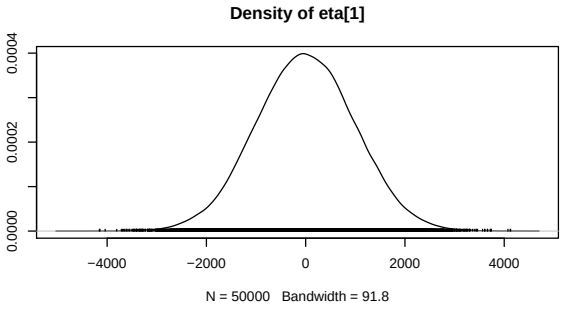

Trace of beta.home[3]
20
10
0
−10
−20
60000
70000
80000
90000
100000
Iterations

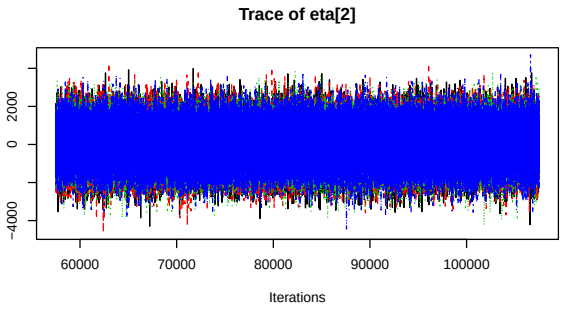

Density of beta.home[3]
0.08
0.06
0.04
0.02
0.00
−20
−10
0
10
20
N = 50000 Bandwidth = 0.4648

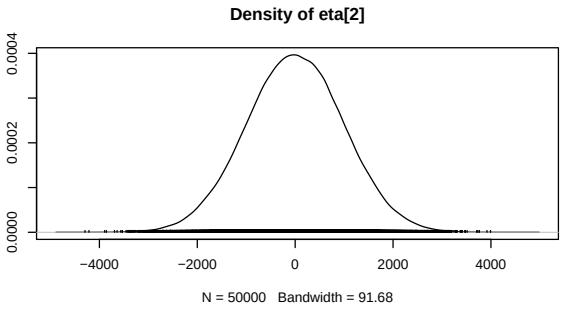

Trace of delta
2000
0
−4000
60000
70000
80000
90000
100000
Iterations

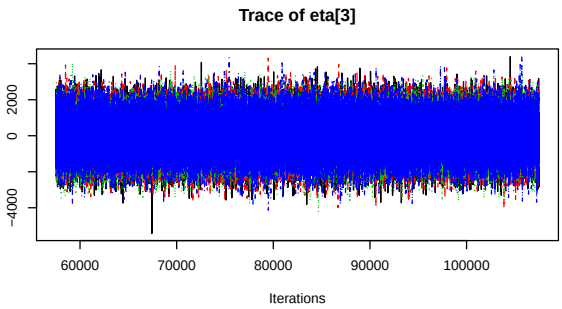

Density of delta
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 50000 Bandwidth = 91.65

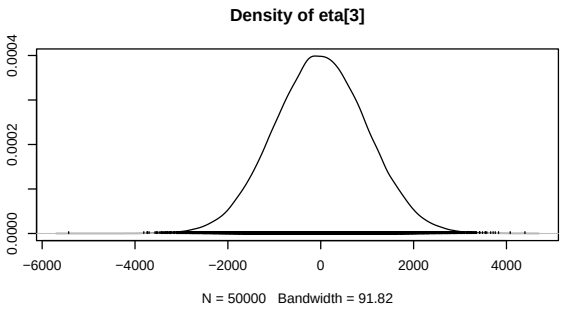

Trace of eta[1]
2000
0
−4000
60000
70000
80000
90000
100000
Iterations

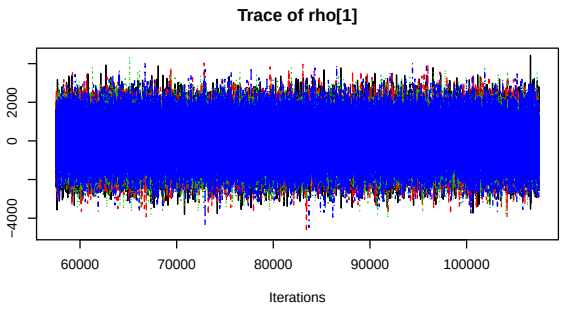

Density of eta[1]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 50000 Bandwidth = 91.8

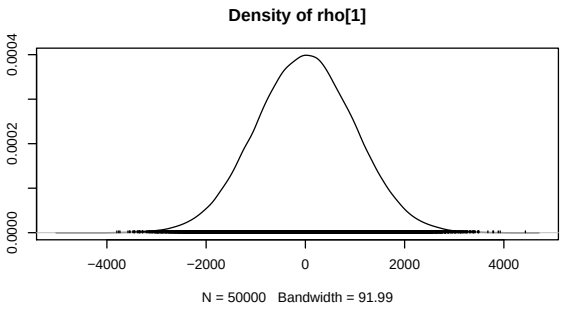

Trace of eta[2]
2000
0
−4000
60000
70000
80000
90000
100000
Iterations

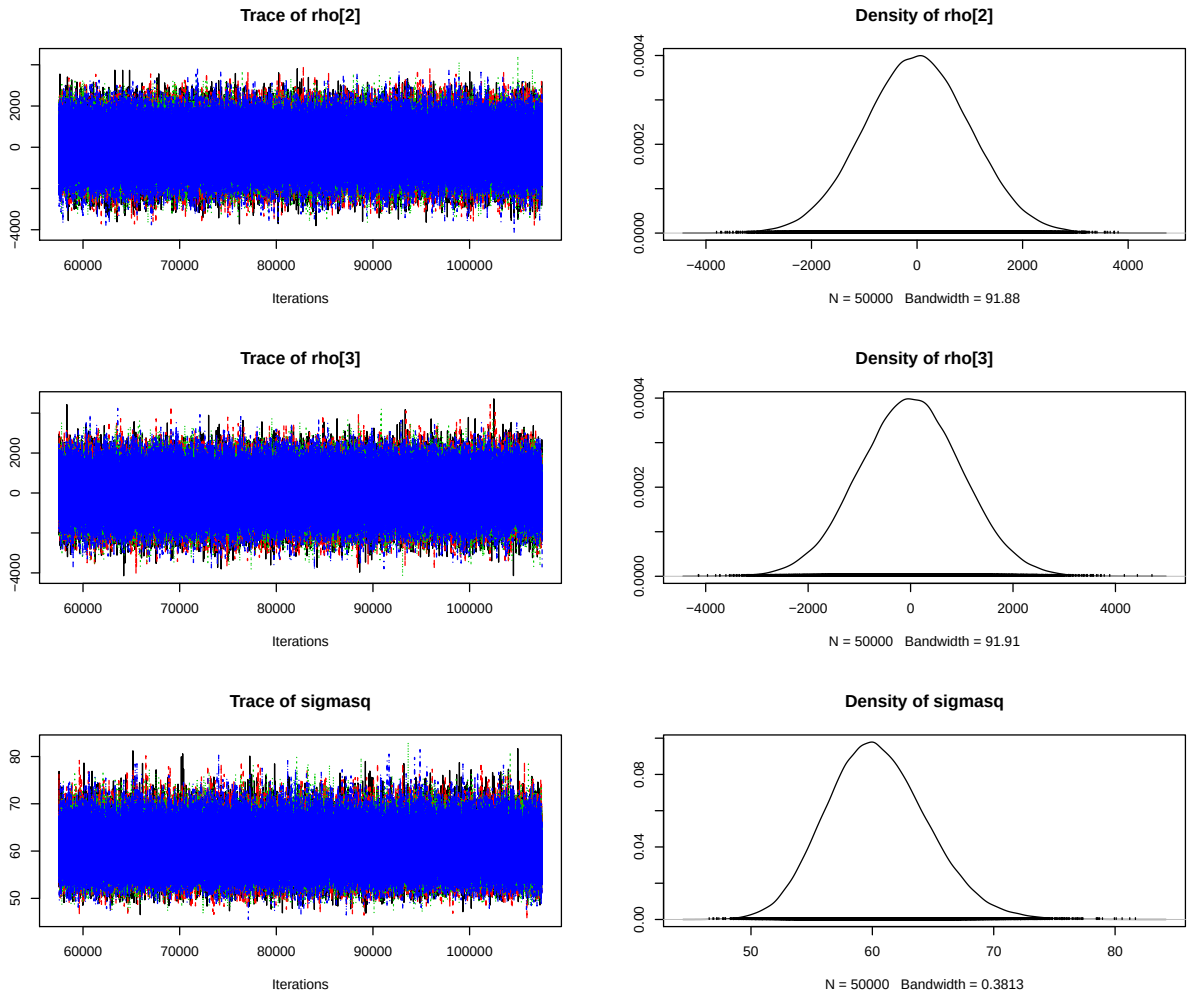

Density of eta[2]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 50000 Bandwidth = 91.68


Trace of eta[3]
2000
0
−4000
60000
70000
80000
90000
100000
Iterations


Density of eta[3]
0.0004
0.0002
0.0000
−6000
−4000
−2000
0
2000
4000
N = 50000 Bandwidth = 91.82


Trace of rho[1]
2000
0
−4000
60000
70000
80000
90000
100000
Iterations


Density of rho[1]
0.0004
0.0002
0.0000
−4000
−2000
0
2000
4000
N = 50000 Bandwidth = 91.99



Converged as `gelman.R.max` = 1.0006 < 1.1 and the plot also looks good.

```
summary(result$coda.sam)
```

```
##
## Iterations = 57501:107500
## Thinning interval = 1
## Number of chains = 4
## Sample size per chain = 50000
##
```

```
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean      SD Naive SE Time-series SE
## beta.away[1] -4.458   2.307 0.005158        0.04314
## beta.away[2] -2.152   2.894 0.006472        0.04389
## beta.away[3]  1.793   3.555 0.007950        0.03792
## beta.defense  0.205   0.124 0.000278        0.00234
## beta.home[1] -3.010   2.256 0.005044        0.04202
## beta.home[2] -1.498   2.701 0.006040        0.04054
## beta.home[3]  0.384   5.037 0.011263        0.04278
## delta         2.725 993.233 2.220936        2.21426
## eta[1]       -2.274 994.867 2.224590        2.22441
## eta[2]       -0.472 993.568 2.221687        2.22170
## eta[3]       -2.563 994.988 2.224861        2.22487
## rho[1]       -3.411 996.831 2.228982        2.22740
## rho[2]        0.128 995.636 2.226310        2.22625
## rho[3]        2.504 995.967 2.227049        2.23786
## sigmasq      60.424   4.140 0.009258        0.00953
##
## 2. Quantiles for each variable:
##
##                   2.5%      25%    50%     75%    97.5%
## beta.away[1]    -8.9266   -6.027 -4.461  -2.916    0.108
## beta.away[2]    -7.7977   -4.108 -2.154  -0.209    3.548
## beta.away[3]    -5.1679   -0.605  1.781   4.192    8.771
## beta.defense    -0.0408    0.122  0.205   0.290    0.446
## beta.home[1]    -7.3885   -4.546 -3.013  -1.502    1.460
## beta.home[2]    -6.7821   -3.324 -1.506   0.313    3.804
## beta.home[3]    -9.4310   -3.019  0.387   3.780   10.283
## delta        -1942.0280 -670.332  3.241 672.268 1953.336
## eta[1]       -1952.3324 -675.372 -4.375 668.658 1945.696
## eta[2]       -1947.0362 -670.343  0.709 669.540 1945.039
## eta[3]       -1943.3195 -674.281 -3.906 668.919 1946.212
## rho[1]       -1960.6739 -675.983 -1.601 669.517 1943.450
## rho[2]       -1952.9968 -671.574  0.900 671.594 1945.814
## rho[3]       -1954.1195 -667.491  3.154 672.737 1962.521
## sigmasq         52.8500   57.560 60.228  63.096   69.031
```

Effective Sample Size

```
(eff.size = effectiveSize(result$coda.sam[, 1:7]))
```

```
## beta.away[1] beta.away[2] beta.away[3] beta.defense beta.home[1] beta.home[2]
##         2859         4352         8808         2808         2881         4444
## beta.home[3]
##        13890
```

The effective sample sizes of all parameters are greater than 400.